cvc5: A Versatile and Industrial-Strength SMT Solver*

Haniel Barbosa[3], Clark Barrett[1], Martin Brain[4], Gereon Kremer[1], Hanna Lachnitt[1], Makai Mann[1], Abdalrhman Mohamed[2], Mudathir Mohamed[2], Aina Niemetz[1(✉)], Andres Nötzli[1], Alex Ozdemir[1], Mathias Preiner[1], Andrew Reynolds[2], Ying Sheng[1], Cesare Tinelli[2], and Yoni Zohar[1,5]

[1] Stanford University, Stanford, USA (✉)niemetz@cs.stanford.edu
[2] The University of Iowa, Iowa City, USA
[3] Universidade Federal de Minas Gerais, Belo Horizonte, Brazil
[4] City, University of London, London, UK
[5] Bar-Ilan University, Ramat Gan, Israel

Abstract. cvc5 is the latest SMT solver in the cooperating validity checker series and builds on the successful code base of CVC4. This paper serves as a comprehensive system description of cvc5's architectural design and highlights the major features and components introduced since CVC4 1.8. We evaluate cvc5's performance on all benchmarks in SMT-LIB and provide a comparison against CVC4 and Z3.

Keywords: automated reasoning · constraint solving · satisfiability modulo theories · cvc5

1 Introduction

SMT solvers are widely recognized as crucial back-end reasoning engines for a variety of applications, including software and hardware verification [19, 52, 60, 68, 82, 86], model checking [41, 42, 98], type checking, static analysis, security [10, 62], automated test-case generation [40, 135], synthesis [2, 65], planning, scheduling, and optimization [127]. Notable SMT solvers include Bitwuzla [92], Boolector [98], CVC4 [21], MathSAT [46], OpenSMT2 [72], SMTInterpol [44], SMT-RAT [50], STP [61], veriT [35], Yices2 [55], and Z3 [90].

Among these, the family of *cooperating validity checker (CVC)* tools [21, 26, 27, 132] have played an important role, both in research and in practice [11, 48, 70, 137, 138]. The most recent incarnation, CVC4, was a from-scratch rewrite of

* This work was supported by AFOSR, AFRL, Amazon Web Services, BSF, Certora, DARPA, ERC, GE Global Research, Google, Intel, Meta, NASA, NSF, ONR, SRC, United Technologies Research Center, and Stanford University—including the Center for Automated Reasoning (Centaur), the Center for Blockchain Research, the Agile Hardware Center (AHA), and the SystemX Alliance. More details can be found at: https://cvc5.github.io/acknowledgements.html.

CVC3, written with the aim of creating a flexible and performant architecture that could last far into the future. The fact that CVC4 has integrated over a decade's worth of SMT research and development while becoming increasingly robust and performance-competitive attests to the success of that endeavor.

In this paper, we introduce cvc5, the next solver in the series. cvc5 is not a rewrite of CVC4 and indeed builds on its successful code base and architecture. Compared to other SMT solvers, cvc5 supports a diverse set of theories (all standard SMT-LIB theories, and many non-standard theories) and features beyond regular SMT solving such as higher-order reasoning and syntax-guided synthesis (SyGuS) [3]. The name-change[6] rather acknowledges both a (mostly) new team of developers as well as the significant evolution the tool has undergone since CVC4 was described in a tool paper published in 2011 [21]. Moreover, cvc5 comes with updated documentation, new and improved APIs, and more user-friendly installation. Most importantly, it introduces several significant new features. Like its predecessors, cvc5 is available under the 3-clause BSD open source license and runs on all major platforms (Linux, macOS, and Windows).

We make the following contributions:

- An in-depth description of the architectural design of cvc5 and how its pieces and modules work together.
- A comprehensive summary of all features that have been added to the solver since CVC4 was introduced in [21].
- A description of major features introduced since CVC4 1.8, the final version of CVC4, including:
  - a new C++ API, and new Python and Java APIs that build on top of it;
  - a new theory solver for the theory of fixed-size bit-vectors;
  - a new and extensive proof-production module;
  - a new procedure for non-linear arithmetic; and
  - a syntax-guided quantifier-instantiation procedure [96].
- Evidence, based on experimental evaluation and industrial use cases, that cvc5 is in fact both *versatile* and *industrial-strength.*

2 Architecture and Core Components

cvc5 supports reasoning about quantifier-free and quantified formulas in a wide range of background theories and their combinations, including all theories standardized in SMT-LIB [22]. It further natively supports several non-standard theories and theory extensions. These include, among others, separation logic, the theory of sequences, the theory of finite sets and relations, and the extension of the theory of reals with transcendental functions.

In this section, we start with a brief overview of the core components of cvc5, and then discuss them in more detail in the following subsections. A high-level overview of the system architecture is given in Figure 1.

[6] Whereas the convention for previous solvers in the CVC family was to use capital letters, here we introduce a new convention of using lower-case letters (or alternatively small capitals, as in this paper, which we find to be more visually appealing).

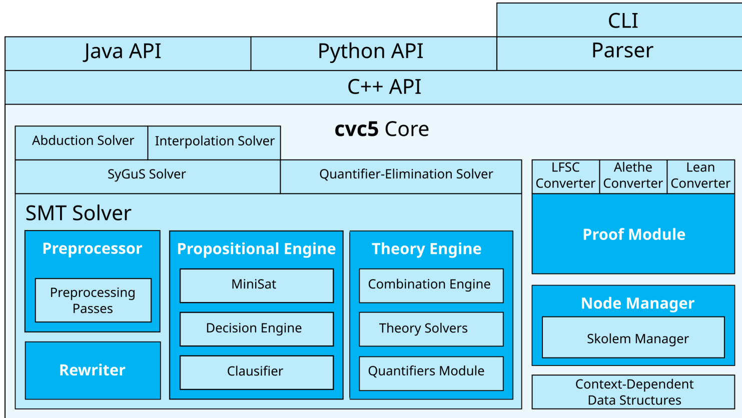

Fig. 1: High-level overview of cvc5's system architecture.

The central engine of cvc5 is the *SMT Solver* module, which is based on the CDCL($\mathcal{T}$) framework [99] and relies on a customized version of the MiniSat propositional solver [57] at its core. The SMT Solver consists of several components: the Rewriter and the Preprocessor modules, which apply simplifications locally (at the term level) and globally (on the whole input formula), respectively; the Propositional Engine, which serves as a manager for the CDCL($\mathcal{T}$) SAT solver; and the Theory Engine, which manages theory combination and all theory-specific and quantified reasoning procedures.

Besides standard satisfiability checking, cvc5 provides additional functionality such as abduction, interpolation, syntax-guided synthesis (SyGuS) [3], and quantifier elimination. Each of these features is implemented as an additional solver built on top of the SMT Solver. The SyGuS Solver is the main entry point for synthesis queries, which encode SyGuS problems as (higher-order) satisfiability problems with both semantic and syntactic constraints [114]. The Quantifier Elimination Solver performs quantifier elimination based on tracking the quantifier instantiations of the SMT Solver [116]. The Abduction Solver and the Interpolation Solver are both SyGuS-based [110] and thus are built as layers on top of the SyGuS Solver.

cvc5 provides a C++ API as the main interface, not just for external client software, but also for its own parser and for additional language bindings in Java and Python. cvc5 also provides a textual command-line interface (CLI), built on top of the parser, which supports SMT-LIBv2 [25], SyGuS2 [104] and TPTP [134] as input languages. The Proof Module can output formal unsatisfiability proofs in three proof formats: Alethe [128], Lean 4 [88], and LFSC [133].

2.1 The SMT Solver Module

The SMT Solver module is the centerpiece of cvc5 and is responsible for handling all SMT queries. Its functionality includes, in addition to satisfiability

checking, constructing models for satisfiable input formulas and extracting assumptions, cores, and proof objects for unsatisfiable formulas. The main components of the SMT Solver module are described below.

Preprocessor. Before any satisfiability check, cvc5 applies to each formula from an input problem a sequence of satisfiability-preserving transformations. We distinguish between (i) required *normalization* passes, e.g., removal of ite terms; (ii) optional *simplification* passes aimed at making the formula easier to solve, e.g., finding entailed theory literals; and (iii) optional *reduction* passes that transform the formula from one logic to another, e.g., from non-linear integer arithmetic to a bit-vector problem with configurable bit-width. Currently, cvc5 implements 34 passes, executed in a fixed order. Optional passes can be enabled and disabled via configuration options. Preprocessing passes are self-contained, and adding or modifying passes does not require knowledge of the internals of the SMT solver engine.

Propositional Engine. The Propositional Engine serves as the core CDCL($\mathcal{T}$) engine [99], which takes the Boolean abstraction of the input formula (together with any lemmas produced during solving) and produces a satisfying assignment for that abstraction. Its main components are the Clausifier and the propositional satisfiability (SAT) solver. The Clausifier converts the Boolean abstraction into Conjunctive Normal Form (CNF), which then serves as input for the SAT solver. In cvc5, as in CVC4, we use a customized version of MiniSat [57] as the core SAT solver. Extensions we have added to MiniSat include: the production of resolution proofs; native support for pushing and popping assertions; and a *Decision Engine* [12], which can be used to create customized decision heuristics for MiniSat.

During its search, the Propositional Engine asserts a theory literal $(\neg)p$ to the Theory Engine as soon as the SAT solver assigns a truth value to the propositional variable abstracting the atom p. We refer to the set of all such literals as the *currently asserted literals*. When checking the consistency of the set L of currently asserted literals in the overall background theory $\mathcal{T}$, we distinguish between two levels of effort: *standard* and *full*, depending on whether the SAT solver has a partial or full model, respectively, for the Boolean abstraction. At standard effort, a theory solver may optionally perform some lightweight consistency checking. At full effort, the theory solver must either produce a lemma (following the splitting-on-demand approach [23]) or determine whether L is satisfiable or not and, in the latter case, produce a *conflict clause*, a clause that is valid in the theory $\mathcal{T}$ but is inconsistent with L.

Rewriter. The Rewriter module is responsible for converting terms via a set of rewrite rules into semantically equivalent normal forms. In contrast to preprocessing, rewriting is done *during solving*. In fact, all major components of cvc5 invoke the Rewriter to ensure that the terms they work with are normalized, thereby simplifying their implementation. Rewrite rules are applied locally, i.e., independent of the currently asserted literals, and are divided into required and optional rules, of which the latter can be enabled or disabled by the user. The Rewriter maintains a cache to avoid processing any term more than once.

Examples of rewrites include simplifications such as $x + 0 \rightsquigarrow x$, normalizations that sort the operands of associative and commutative operators, and operator eliminations such as $x \leq y \rightsquigarrow y + 1 > x$ (when x and y have integer sort). In certain contexts, e.g., enumerative SyGuS approaches, aggressive rewriting rules, which would be detrimental to SMT solving, can be beneficial. Such rules are implemented in an *Extended Rewriter*, which is enabled when needed.

To help automate improvements to the Rewriter, we developed a workflow that detects and enumerates new rewrite rule candidates using the SyGuS solver [101]. It works by detecting and suggesting *critical pairs*, i.e., pairs of equivalent terms that are not rewritten to the same term by the current rules.

Theory Engine. The Theory Engine is the main entry point for checking the theory consistency of the theory literals asserted by the Propositional Engine. It dispatches each of these literals to the appropriate theory solvers and is further responsible for dispatching any propagated literals or lemmas generated by the theory solvers back to the Propositional Engine.

When multiple theory solvers are enabled, the Combination Engine submodule is responsible for coordinating between them. Like CVC4, cvc5 uses the *polite* theory combination mechanism [74,108,130]. This includes propagating or performing case splits on equalities and disequalities between *shared* terms (terms appearing in the literals of more than one theory solver). As in CVC4, the algorithm for computing these splits is based on care graphs [75].

The Combination Engine controls the Model Manager, which is responsible for combining models from multiple theories and constructs a model for the input formula. The Model Manager also maintains an equivalence relation E over all the terms in the input formula, induced by all of the currently asserted literals that are equalities. When invoked, the Model Manager has the responsibility of assigning concrete values to each equivalence class of E with the assistance of the individual theory solvers, which provide values for terms in their theory. Typically, the Model Manager is invoked only when the theory solvers have reached a saturation point that allows the Theory Engine to conclude that the input problem is satisfiable (and thus, a model can be constructed successfully).

As in CVC4, each sub-formula of the input that starts with a quantifier is abstracted by a propositional variable. When any such variable or its negation is asserted, the Theory Engine dispatches the corresponding quantified formula to the Quantifiers Module, which generates suitable quantifier instantiations. Since certain techniques for handling quantified formulas, e.g., E-matching [89], require knowledge of the state and terms known by the other theory solvers, this module has access to *all* equality information from all theory solvers.

Theory Solvers. cvc5 supports a wide range of theories, including all theories standardized in SMT-LIB. Each theory solver relies on an Equality Engine Module, which implements congruence closure over a configurable set of operators, typically those that belong to the solver's theory. The Equality Engine is responsible for quickly detecting conflicts due to equality reasoning. In addition, all theories communicate reasoning steps to the rest of the system via the Theory Inference Manager. Every theory solver emits lemmas, conflict clauses,

and propagated literals through this interface. The Theory Inference Manager implements or simplifies common usage pattern like caching and rewriting lemmas, proof construction, and collection of statistics. Every lemma or conflict sent from a theory is associated with a unique identifier for its kind, the *inference identifier*, which is a crucial debugging aid. Below, we briefly survey the theory solvers in cvc5, along with their main implementation techniques.

Linear Arithmetic. The linear arithmetic solver [78] extends the simplex procedure adapted for SMT by Dutertre and de Moura [56]. It implements a sum-of-infeasibilities-based heuristic [79], an integration with the external GLPK LP solver [80], and certain heuristics proposed by Griggio [63]. Integer problems are handled by solving their real relaxation before using branching [64] and cutting planes [54] to find integer solutions. The branch-and-bound method optionally generates lemmas consisting of ternary clauses inspired by *unit-cube* tests [39].

Non-linear Arithmetic. For non-linear arithmetic problems, cvc5 resorts to linear abstraction and refinement. It uses a combination of independent sub-solvers integrated with the linear arithmetic solver and invoked only when the linear abstraction is satisfiable. One sub-solver implements cylindrical algebraic coverings [1], while the other sub-solvers are based on incremental linearization [45]. A variety of lemma schemas are used to assert properties of non-linear functions (e.g., multiplication and trigonometric functions) in a counterexample-guided fashion [123]. Non-linear integer problems are solved by incremental linearization and incomplete techniques based on reductions to bit-vectors.

Arrays. As in CVC4, the array solver is based on a decision procedure by de Moura and Bjørner [91] but following the more detailed description by Jovanović and Barrett [75]. An alternative experimental implementation based on an approach by Christ and Hoenicke [43] is also available.

Bit-Vectors. For the theory of fixed-size bit-vectors, cvc5's main approach is *bit-blasting*, which refers to the process of translating bit-vector problems into equisatisfiable SAT problems, and is applied after preprocessing. In cvc5, we distinguish two modes for bit-blasting: *lazy* and *eager*. Lazy bit-blasting seamlessly integrates with the CDCL($\mathcal{T}$) infrastructure of cvc5 and fully supports the combination of bit-vectors with any theory supported by cvc5. It further leverages the full power of cvc5's Equality Engine for reasoning about equalities over bit-vector terms and also uses the solve-under-assumptions feature [57] supported by many state-of-the-art SAT solvers. For problems that can be fully reduced to bit-vectors, cvc5 can also be used in eager mode. This mode does not rely on solving under assumptions, but instead directly asserts all of the bit-blasted constraints to the SAT solver, which usually enables more simplifications. Additionally, cvc5 supports the Ackermannization and eager bit-blasting of constraints involving uninterpreted functions and sorts [66].

Datatypes. For quantifier-free constraints over datatypes, we use a rule-based procedure that follows calculi already implemented in CVC4 [24, 112] and that optimizes the sharing of selectors over multiple constructors [125].

Floating-Point Arithmetic. Formulas in the theory of floating-point arithmetic are translated to equisatisfiable formulas in the theory of bit-vectors, in a process referred to as *word-blasting*. For this, cvc5 integrates the SymFPU [37] library, which was first used in CVC4 and has also been integrated in the Bitwuzla SMT solver [92]. This approach admits several optimizations compared to earlier solvers, which translate directly to the *bit-level*, e.g., CNF or AIGs. Another difference from older approaches [38] is that translation is done at the formula level instead of the term level. Conversions between real and floating-point terms are treated as uninterpreted functions and refined if the models of the real arithmetic and the floating-point solver do not agree. The refinement lemmas use the monotonicity of the conversion functions to constrain the floating-point and real arithmetic terms to matching intervals that exclude the current model.

Sets and Relations. cvc5 implements a solver for the parametric theory of finite sets, i.e., sets whose elements are of any sort supported by cvc5. The core decision procedure for sets is extended with support for cardinality constraints [13]. The set theory solver is extended with a sub-module that specializes in relational constraints [87], where relations are modeled as sets of tuples.

Separation Logic. In separation logic, the semantics of constraints assume a location and data type for specifying the model of the heap. cvc5 supports an extension of the SMT-LIB language for separation logic [73], in which the location and data types of the heap can be any sort supported by cvc5. The classical separation logic connectives are treated as theory predicates which are lazily reduced to constraints over sets and uninterpreted functions [115].

Strings and Sequences. For strings and sequences, cvc5 implements a solver consisting of multiple layered components. At its core, the solver reasons about length constraints and word equations [84], supplemented with reasoning about code points to handle conversions between strings and integers efficiently [119]. Extended functions such as string replacement are lazily reduced to word equations after context-dependent simplifications [126]. When necessary, the regular expressions in input problems are unfolded and derivatives are computed [85]. The string theory solver further incorporates aggressive simplification rules that rely on abstractions to derive facts about string terms [118]. Finally, conflicts are detected eagerly on partial assignments from the SAT solver by computing the congruence closure and constant prefixes and suffixes of string terms.

Uninterpreted Functions. The theory of uninterpreted functions is handled in largely the same way as in CVC4. It follows Simplify's approach [53] extended with support for fixed finite cardinality constraints [121]. This extension is used in combination with finite-model-finding techniques for finding finite models based on minimal interpretations of uninterpreted sorts.

Quantifiers. Quantified formulas are all handled by the Quantifiers Module, which resembles a theory solver. The module contains many sub-solvers, all based on some form of quantifier instantiation, and each specializing in solving specific classes of quantified formulas. The Quantifiers Module relies on heuristic E-matching when uninterpreted functions are present [89]. This technique

is supplemented by conflict-based instantiation for detecting when an instantiation is in conflict with the currently asserted literals [16, 124]. The Quantifiers Module additionally incorporates finite-model-finding techniques, which are useful for detecting satisfiable input problems [122]. It also relies on enumerative approaches when other techniques are incomplete [109]. For quantifiers over linear arithmetic, it uses a specialized counterexample-guided based approach for quantifier instantiation [116]. An extension of this technique is used for quantified bit-vector logics [95]. For other quantified logics in pure background theories, e.g., over floating-point or non-linear arithmetic, cvc5 relies on syntax-guided quantifier instantiation [96]. The Quantifiers Module also contains sub-solvers implementing more advanced solving paradigms, including: a module for doing Skolemization with inductive strengthening and enumeration of sub-goals for inductive theorem proving problems [117], a finite-model-finding technique for recursive functions [113], and a solver for syntax-guided synthesis [114].

2.2 Proof Module

The Proof Module of cvc5 was built from scratch and replaces the proof system of CVC4 [67, 77], which was incomplete and suffered from a number of architectural shortcomings. The design of cvc5's proof module was guided by the following principles. First, the overhead incurred by proof production should be at most linear in the solving time. Second, the emitted proofs should be detailed enough to enable efficient (i.e., polynomial) checking, ensuring that proof checking is inherently simpler than solving. Third, disabling a system component when in proof production mode because it lacks adequate proof generation capabilities should be done rarely and only if the component is not crucial for performance. Finally, given the different needs of users and the trade-offs offered by different proof systems, proof production should be flexible enough to allow the emission of proofs in different formats.

Following these design principles, the Proof Module in cvc5 produces detailed proofs for nearly all of its theories, rewrite rules, preprocessing passes, internal SAT solvers, and theory combination engines. It further supports eager and lazy proof production with built-in proof reconstruction. This enables proof production for some notoriously challenging functionalities, such as substitution and rewriting (common, for example, in simplification under global assumptions and in string solving [126]). Furthermore, although it maintains internally a single proof representation, cvc5 is able to emit proofs in multiple formats, including those supported by the LFSC [133] proof checker and the Lean 4 [88], Isabelle/HOL [100] and Coq [30] proof assistants.

2.3 Node Manager

Formulas and terms are represented uniformly in cvc5 as nodes in a directed acyclic graph, reference-counted and managed by the Node Manager. The Node Manager further maintains a Skolem Manager, which is responsible for tracking

Skolem symbols introduced during solving. All cvc5 instances in the same thread share the same Node Manager instance.

Nodes are immutable and are aggressively shared using *hash consing*: whenever a new node is about to be created, the Node Manager checks whether a node with the same structure already exists, and if it does, it returns a reference to the existing node instead. Besides saving memory, this ensures that syntactic equality checks can be performed in constant time (by comparing the unique ids assigned to each node). Reference counting allows the Node Manager to determine when to dispose of nodes. Weak references are used whenever possible to limit the overhead of reference counting.

Nodes store 96 bits of metadata (id, reference count, kind, and number of children) and a variable number of pointers to child nodes. The kind of a node can be an operator kind, e.g., addition, or a leaf kind, e.g., a variable. Optional additional static information associated with nodes can be stored separately in hash maps referred to as *node attributes*. Since node attributes are managed by the Node Manager, which may be shared by multiple solver instances, attributes must only be used to capture inherent node properties (i.e., properties that are independent of run-time options).

Many theory solvers, including those for quantifiers, strings, arrays, nonlinear arithmetic, and sets, introduce terms with Skolem (i.e., fresh) constants during solving. Such constants are centrally generated by the Skolem Manager, which also associates with each of them a term of the same sort, the constant's *witness form*. If the computed witness form for a constant matches that of a previously used constant, the previous constant can be reused. This not only provides a deterministic way of generating fresh constants during solving but also allows the system to minimize the number of introduced constants. This reuse is crucial for performance in some theory solvers [120].

2.4 Context-Dependent Data Structures

Certain applications of SMT solvers require multiple satisfiability checks with similar assertions. To support such applications, the SMT-LIB standard includes commands to save (with a *push* command) the current set of user-level assertions and restore (with a *pop* command) a previous set. This allows the solver to reuse parts of the work from earlier satisfiability checks and amortizes startup cost. Most of the state of cvc5 depends directly or indirectly on the current set of assertions. So whenever the user pushes or pops, cvc5 has to save or restore the corresponding state. Similarly, whenever the SAT solver makes a decision or backtracks to a previous decision point, each theory solver has to save or restore the corresponding information.

To support these operations, cvc5 defines a notion of *context level*, which increases with each push and decreases with each pop operation, and implements *context-dependent data structures*. These data structures behave similarly to corresponding mutable data structures provided in the C++ standard library, except that they are associated with a context level and automatically save and restore their state as the context increases or decreases. For efficiency reasons,

```
s = Solver()
i = s.getIntegerSort()
x = s.mkConst(i, "x")
s.assertFormula(
  s.mkTerm(kinds.Equal,
    s.mkTerm(kinds.Mult,
      x, s.mkInteger(2)),
    s.mkInteger(4)))
s.checkSat()
```

(a) The base cvc5 Python API

```
solve(2 * Int("x") == 4)
```

(b) The "pythonic" API

Fig. 2: Example of using the Python APIs of cvc5.

this state data is stored using a region-based custom allocator that allocates one region per context level, allowing all state data associated with a level to be freed simultaneously by simply freeing the corresponding region.

3 Highlighted Features

In this section, we discuss features that are new in cvc5 as well as some of the more prominent user- and developer-facing features. We compare them to their counterparts in CVC4 when applicable.

Application Programming Interfaces (APIs). cvc5 provides a lean, comprehensive, and feature-complete C++ API, which also serves as the main interface for the parser module and the basis for all other language bindings. The parser module uses the same API as external users, without any special privileges. cvc5's C++ API has been designed and written from scratch and thus is not backwards compatible with CVC4's C++ API. It is centered around the `Solver` class, which represents a cvc5 instance and implements methods for tasks such as creating terms, asserting formulas, and issuing checks.

cvc5's Python API is built on top of cvc5's C++ API using Cython [29] and makes all of cvc5's features accessible to Python users. It is a straightforward translation of the C++ API without added syntactic sugar such as operator overloading. Additionally, however, cvc5 provides a higher-level layer on top of its Python API, which is more user-friendly and *pythonic*. This layer provides automatic solver management, allows SMT terms to be constructed using Python infix operators, and converts Python objects to SMT terms of the appropriate sort. This leads to much more succinct code, as shown in Figure 2, which compares using the high- and low-level Python APIs to solve the integer equation $2 \cdot x = 4$. The higher-level Python API is based on and designed to work as a drop-in replacement for Z3py, the Python API of Z3 [90].

cvc5's Java API is implemented via the Java Native Interface (JNI), which allows Java applications to invoke native code and vice versa [83]. In contrast, CVC4 uses SWIG [28] to semi-automatically generate bindings. One of the challenges of developing a Java API, and the main motivation for implementing it

manually instead of using SWIG, is the interaction between Java's garbage collector and cvc5's reference-counting mechanism for terms and sorts. The new API implements the AutoCloseable interface to destroy the underlying C++ objects in the expected order. It mostly mirrors the C++ API and supports operator overloading, iterators, and exceptions. There are a few differences from the C++ API, such as using arbitrary-precision integer pairs, specifically, pairs of Java BigInteger objects, to represent rational numbers. In contrast to the old Java API, the new API puts greater emphasis on using Java-native types such as List<T> instead of wrapper classes for C++ types such as std::vector<T>.

Documentation. We provide comprehensive documentation for both cvc5 users [8] and developers [6]. User documentation contains instructions for building and installing cvc5 and its dependencies, extensive documentation and examples of common uses cases for all available APIs, and a thorough description of all supported non-standard theories with examples. Developer documentation provides details of cvc5 internals and instructions for contributions, including guidelines for coding and testing, and a recommended development workflow.

Proofs. As mentioned above, cvc5 has a new proof system. Proofs are stored internally using a new custom intermediate representation. Multiple output proof formats are supported via target-specific post-processing transformations on this internal representation. The final proof object can then be pretty-printed and saved in a text file. The currently supported output proof formats include LFSC [133], Alethe [128], and the language of the Lean 4 [88] proof assistant.

CVC4 proofs exclusively used the LFSC format. cvc5 continues support for LFSC but with a new, more user-friendly syntax. LFSC is a logical framework, based on Edinburgh LF [69], which was explicitly designed to facilitate the production and checking of fine-grained proofs in SMT. It comes with a small and high-performance proof checker, which is generic in the sense that it takes as input both a proof term p and a *proof signature*, a definition of the data types and proof rules used to construct p. The checker verifies that p is well-formed with respect to the provided signature. We have defined proof signatures for all the individual theories supported by cvc5. These definitions can be combined together as needed to define a proof system for any combination of those theories. When emitting proofs in LFSC, cvc5 includes all the relevant signatures as a preamble to the proof term.

The Alethe proof format is a flexible proof format for SMT solvers based on SMT-LIB. It includes both coarse- and fine-grained steps and was first implemented in the veriT solver [34]. Alethe proofs can be checked via reconstruction within Isabelle/HOL [15,129] as well as within Coq, the latter via the SMTCoq plugin [5,58]. Our main motivation for producing Alethe proofs is to leverage these proof reconstruction infrastructures, thus enabling the trustworthy integration of cvc5 in Isabelle/HOL and Coq. Users of these tools can leverage the integration to dispatch selected goals to cvc5 for proving, thereby increasing the level of automation available to them without requiring a larger trusted core. These integrations represent ongoing work in cvc5 and are being carried out in close collaboration with both Isabelle/HOL and Coq experts.

Although we aim to have a similar full integration in the Lean 4 [88] proof assistant in the future, cvc5 currently only supports the use of Lean 4 as an external checker; i.e., cvc5 can emit proofs as Lean terms (for a subset of the theories supported by cvc5), and Lean 4 can then check these proofs. Since the underlying logic of Lean 4 is an extension of that of LFSC, this functionality follows an approach similar to that used for LFSC by modeling cvc5 proof rules as Lean types and reducing proof checking to type checking.

Syntax-Guided Synthesis. cvc5 has native support for syntax-guided synthesis (SyGuS) problems [3]. As mentioned, the cvc5 core has a dedicated module for encoding SyGuS problems into (higher-order) SMT formulas, annotated with syntactic restrictions. These restrictions are represented via a deep embedding into the theory of datatypes. Internally, after encoding the SyGuS problem, a sub-module of the quantifiers theory, called the synthesis engine, is the main entry point for solving. Based on the shape of the input, it uses one of three approaches. If the input problem has no syntactic restrictions, and is in *single invocation* form [114], that is, all functions to synthesize are applied to the same argument list, then it uses a quantifier-instantiation based approach. Otherwise, it uses one of two enumerative approaches, depending on the properties of the input [111]. The SyGuS solver also implements further refinements and extensions of the enumerative approaches, including algorithms for decision-tree learning [4] for programming-by-example problems, extended rewriting for enumeration [101], piecewise-independent unification [17], and static grammar-reduction techniques. Furthermore, the SyGuS solver contains specialized procedures to support an efficient implementation of interpolation and abduction.

Interpolation and Abduction. cvc5 computes abducts and Craig interpolants [51] using solvers built on top of the SyGuS solver. The solver for interpolation translates an interpolation query into a SyGuS conjecture whose solutions are interpolants. Specifically, given quantifier-free formulas A and C over any combination of the theories supported by cvc5, the interpolation solver solves for B in the SyGuS conjecture $A \rightarrow B \ \wedge \ B \rightarrow C$, with the syntactic restriction that B's free symbols range over the symbols shared by A and C. Any synthesized solution for B is, by construction, a Craig interpolant for A and C.

Abduction is the process of constructing a formula B that is enough to add to a formula A to prove some goal formula C (equivalently, to make the formula $F = A \wedge B \wedge \neg C$ unsatisfiable). cvc5's abduction solver reduces this problem to a SyGuS one where C is the formula to be synthesized and F is the semantic constraint. Optionally, the user can also impose syntactic restrictions on the abduct B. The SyGuS solver implements specific optimizations for abduction queries, such as using unsat cores to prune classes of invalid candidate solutions [110].

Non-Linear Arithmetic. The new sub-solver for non-linear arithmetic is based on cylindrical algebraic coverings and closely follows [1], with some notable extensions. The implementation uses the libpoly library [76], which provides polynomial arithmetic and most algebraic routines required for the computation of cylindrical algebraic decompositions and coverings. Infeasible subsets are computed by tracking all contributing assertions for every covering. The infeasible

subset is then obtained from the union of assertions from the top-level covering. The sub-solver implements several different variable orderings, as these can have a significant impact on run-times in practice. Apart from classical variable orderings used for cylindrical algebraic decomposition, some experimental orderings based on machine learning have been implemented, roughly following ideas from England et al. [59]. (Mixed real-) integer problems are supported by dynamically injecting intervals into coverings to cover gaps that do not contain integers.

Higher-Order Logic. cvc5 has been extended with partial support for higher-order logic [18]. The extension is based on a pragmatic approach in which λ-abstractions are eliminated eagerly via lambda lifting [71]. This approach is used with the theory solver for the quantifier-free fragment of the theory of equality with uninterpreted functions (EUF) and with the quantifier-instantiation technique based on E-matching with triggers [53, 89]. For the EUF solver, we added support for (dis)equality constraints between functions, via an extensionality inference rule, and for partial applications of (Curried) functions. For quantifier instantiation, we modified several of the data structures for E-matching to incorporate matching in the presence of equalities between function values, function variables, and partial function applications. The extension also uses custom axioms, such as an axiom simulating how functions are updated, to improve the generation of new λ-abstractions, since cvc5 does not yet perform HO-unification, which would allow it to synthesize arbitrary λ-abstractions.

New Bit-Vector Solver. cvc5 features a new bit-blasting solver, which supports the use of off-the-shelf SAT solvers such as CaDiCaL [31] or CryptoMiniSat [131] as SAT back-ends for *both* the eager and lazy bit-blasting approaches. In contrast, CVC4's lazy bit-blasting solver relied on a customized version of MiniSat and did not allow the use of more recent state-of-the-art SAT solvers.

Int-Blasting. In addition to bit-blasting, cvc5 implements *int-blasting* techniques, which reduce bit-vector problems to equisatisfiable non-linear integer arithmetic problems [97, 138]. These techniques are orthogonal to bit-blasting and especially effective on unsatisfiable formulas over large bit-widths.

Syntax-Guided Quantifier Instantiation. cvc5 features a new theory-agnostic enumerative quantifier-instantiation technique we call *syntax-guided quantifier instantiation* [96]. This technique leverages cvc5's SyGuS solver to synthesize terms for quantifier instantiation in a counterexample-guided manner.

Unsatisfiable Cores. In cvc5, unsat (short for unsatisfiable) core extraction has been completely overhauled. It now uses the new proof infrastructure for tracking preprocessing transformations, which, differently from CVC4's, supports most of the preprocessing passes. Unsat cores can be extracted based on the constructed proof or via the tracked preprocessing and assumption-based unsat core extraction [47]. For the latter, cvc5 uses the solve-under-assumptions feature available in the MiniSat-based SAT engine. This is a lightweight solution that does not require the generation of proofs in the SAT solver and full preprocessing proofs. However, if a user requests both unsat cores and proofs, cvc5 switches to proof-based unsat core extraction using the new proof infrastructure.

Distributed and Central Policies for Equality Reasoning. As mentioned in Section 2, the Combination Engine manages theory combination, and theory solvers manage their interactions with the rest of the system via their Equality Engine. In contrast to CVC4, the policy for assigning an Equalitiy Engine to a theory solver in cvc5 is configurable. In the *distributed* policy, a new Equality Engine is generated and assigned for each theory solver. These theory solvers perform congruence closure and their theory-specific reasoning locally. The advantage of this approach is that the constraints are local to the theory and thus do not lead to overhead when combined with other theories. In the *central* policy, a single, shared Equality Engine is assigned to all theory solvers. The advantage of this approach is that communication of facts between theory solvers happens automatically, which in turn can trigger theory propagations more eagerly. Both policies use the same core Equality Engine Module. Each theory solver has been refactored to be agnostic with respect to the equality policy.

Decision Heuristic. For Boolean reasoning, in addition to MiniSat's decision heuristic, cvc5 implements a separate decision heuristic which uses the original Boolean structure of the input to keep track of the *justified* parts of the input constraints, i.e., the parts where it can infer the value of terms based on a (partial) assignment to sub-terms. To make decisions, this new heuristic traverses assertions not satisfied by the currently asserted literals, computing the desired values (starting with true as the desired value for the root) for each term until it finds an unasserted literal that would contribute towards a desired value. This heuristic is a reimplementation and extension of a heuristic [12] implemented in CVC4. The heuristic optionally prioritizes assertions that most frequently contributed to conflicts in the past using a dynamic ordering scheme.

Additional Features. Many more aspects and features have been improved and implemented with the goal of providing useful information to users and developers. Notable examples include: a complete overhaul of CVC4's mechanism for collecting statistics; improved bookkeeping for information about theory lemmas; and a general mechanism for communicating additional information to users such as quantifier instantiations and terms enumerated by the SyGuS solver.

4 Evaluation

We evaluate cvc5's overall performance (commit `5f998504`) by comparing it against Z3 4.8.12 [90] and CVC4 1.8.[7] Z3 is a widely used, high-performance SMT solver which, like cvc5, supports a wide range of theories. We compare against CVC4 to illustrate some of the performance improvements implemented as part of the move to cvc5. To run CVC4 optimally, we use the same command-line options as those in CVC4's competition script for SMT-COMP 2020 [9]. Similarly, for cvc5, we use a (slightly updated) version of the competition script from SMT-COMP 2021 [7]. For some logics, e.g., quantified logics, these scripts try multiple options in a sequential portfolio.

[7] The artifact of this evaluation is archived in the Zenodo open-access repository [14].

Division	cvc5	CVC4	Z3
Arith (7104)	6593	6498	**6844**
Bitvec (6045)	**5741**	5690	5664
Equality (12159)	6677	**6681**	4688
Equality+LinearArith (55948)	49395	48487	**49503**
Equality+MachineArith (4712)	**2065**	1832	1804
Equality+NonLinearArith (17260)	**11088**	10906	9341
FPArith (3170)	**2625**	2113	2593
QF Bitvec (42450)	**41569**	41448	40582
QF Equality (16254)	**16124**	16121	16115
QF Equality+Bitvec (16518)	16274	**16333**	16318
QF Equality+LinearArith (3924)	3778	3782	**3822**
QF Equality+NonLinearArith (673)	598	610	**616**
QF FPArith (76084)	**75998**	75965	75816
QF LinearIntArith (9765)	8619	**8778**	8464
QF LinearRealArith (2008)	1849	**1881**	1864
QF NonLinearIntArith (24261)	17525	16860	**18357**
QF NonLinearRealArith (11552)	**10889**	9207	10354
QF Strings (69863)	69231	**69367**	68074
Total (379750)	**346638**	342559	340819

Table 1: Benchmarks solved by cvc5, CVC4, and Z3 with a 20 minute time limit.

We ran all experiments on a cluster equipped with Intel Xeon E5-2620 v4 CPUs. We allocated one CPU core and 8GB of RAM for each solver and benchmark pair and ran each benchmark with a 20 minute time limit, the same time limit used at SMT-COMP 2021 [102]. We used all non-incremental SMT-LIB [22] benchmarks for our evaluation, with the exception of 45 (misclassified) benchmarks that have quantifiers in quantifier-free logics and 1128 (misclassified) benchmarks that have non-linear literals in linear arithmetic logics. These are known misclassifications in the current release of SMT-LIB. Note that many benchmarks in SMT-LIB come from industrial applications.

Table 1 shows the number of solved benchmarks for each solver using the same divisions as those used for SMT-COMP 2021. There were no disagreements among the solvers on the satisfiability of benchmarks. Overall, cvc5 solves the largest number of benchmarks. Compared to CVC4, cvc5 solves fewer benchmarks in the quantifier-free linear integer arithmetic division due to refactorings related to adding proof support. In the quantifier-free equality and bit-vector division, cvc5 also solves fewer benchmarks, which we attribute to the fact that the new bit-vector solver has not yet been optimized for theory combination. Finally, for quantifier-free string benchmarks, there have been bug fixes since CVC4 that affected performance.

In addition to regularly participating in SMT-COMP, cvc5 and CVC4 also participate in the CADE ATP System Competition (CASC) and in SyGuS-Comp [103]. In CASC, cvc5 tends to perform in the middle of the pack on untyped theorem divisions (unsatisfiable quantified UF in SMT-LIB parlance), and towards the top of the pack on theorems with arithmetic. The last time SyGuS-Comp was held was in 2019, when CVC4 won four out of five tracks.

CVC4 is used extensively in industry, and our users are in the process of updating to cvc5. Examples of its use include: a back-end for Zelkova, a tool developed at Amazon to reason about AWS Access Policies [10, 11, 33]; a back-end for Boogie [20], which is used in many projects including Dafny [81] and the Move Prover [137], a tool used to formally verify smart contracts; a back-end at Certora, another company engaged in formal verification of smart contracts [138]; a back-end for Sledgehammer [32], a tool for discharging proof obligations in Isabelle used by Isabelle's own industrial users; and a back-end for SPARK [70], a development environment for safety-critical Ada programs.

5 Future Work

We briefly highlight a few current development directions for cvc5.

Optimization Solver. Optimization modulo theories (OMT) [136] is an extension of SMT, which requires a solver not only to determine satisfiability but also to return a satisfying assignment (if any) that optimizes one or more objectives. OMT is already supported by several solvers including MathSAT [46] and Z3. cvc5 already has internal infrastructure for supporting OMT queries. We aim to improve and expose (through the APIs) this capability in the near future.

Theory of Bags. cvc5 has preliminary support for a theory of multisets (or *bags*) that can be implemented via a reduction to linear integer arithmetic [107]. We plan to extend this theory with higher-order combinators such as map and fold. With these combinators, and encoding relational tables as bags of tuples, cvc5 will be able to support several commonly-used table operations, with the goal of facilitating reasoning about SQL queries and database applications.

Floating-Point Arithmetic. In addition to word-blasting, we plan to leverage our work on invertibility conditions [36] to lift the local search approach for bit-vectors from [93, 94] to floating-point arithmetic.

Internal Portfolio. Due to the computational complexity of SMT, there is often no single strategy that works best for all problems. As a result, users of SMT solvers often rely on portfolio approaches to try different sets of options, either in parallel or sequentially, as we did in Section 4. Implementing portfolio approaches that use the solver as a black box is sub-optimal because some work, such as parsing, has to be duplicated. The cvc5 roadmap includes plans to support portfolio solving internally, thereby avoiding that additional overhead. We further plan to provide predefined portfolios tuned for specific use cases. As one example of the different needs of different use cases, some applications prefer the solver to always return quickly (even if the answer is "unknown") whereas others expect the solver to try as hard as possible to solve a given problem.

New Parser. cvc5's current parser is inherited from CVC4 and is based on the ANTLR 3 parser generator [105]. In addition to relying on a now deprecated version of ANTLR, the parser is unacceptably slow on large inputs and provides no API for user applications to interact with. A new parser using Flex [106] and Bison [49] is in development. The new parser will also provide an API allowing users to parse whole files or individual terms.

References

1. Ábrahám, E., Davenport, J.H., England, M., Kremer, G.: Deciding the consistency of non-linear real arithmetic constraints with a conflict driven search using cylindrical algebraic coverings. J. Log. Algebraic Methods Program. **119**, 100633 (2021). https://doi.org/10.1016/j.jlamp.2020.100633
2. Alur, R., Bodík, R., Juniwal, G., Martin, M.M.K., Raghothaman, M., Seshia, S.A., Singh, R., Solar-Lezama, A., Torlak, E., Udupa, A.: Syntax-guided synthesis. In: Formal Methods in Computer-Aided Design, FMCAD 2013, Portland, OR, USA, October 20-23, 2013. pp. 1–8. IEEE (2013), https://ieeexplore.ieee.org/document/6679385/
3. Alur, R., Bodík, R., Juniwal, G., Martin, M.M.K., Raghothaman, M., Seshia, S.A., Singh, R., Solar-Lezama, A., Torlak, E., Udupa, A.: Syntax-guided synthesis. In: Formal Methods in Computer-Aided Design, FMCAD 2013, Portland, OR, USA, October 20-23, 2013. pp. 1–8. IEEE (2013), http://ieeexplore.ieee.org/document/6679385/
4. Alur, R., Radhakrishna, A., Udupa, A.: Scaling enumerative program synthesis via divide and conquer. In: Legay, A., Margaria, T. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 23rd International Conference, TACAS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10205, pp. 319–336 (2017). https://doi.org/10.1007/978-3-662-54577-5_18
5. Armand, M., Faure, G., Grégoire, B., Keller, C., Théry, L., Werner, B.: A modular integration of SAT/SMT solvers to Coq through proof witnesses. In: Jouannaud, J.P., Shao, Z. (eds.) Certified Programs and Proofs. Lecture Notes in Computer Science, vol. 7086, pp. 135–150. Springer (2011). https://doi.org/10.1007/978-3-642-25379-9_12
6. cvc5 Authors: cvc5 developer documentation. https://github.com/cvc5/cvc5/wiki (2021)
7. cvc5 Authors: cvc5 SMT-COMP 2021 Single Query run script. https://github.com/cvc5/cvc5/blob/smtcomp2021/contrib/competitions/smt-comp/run-script-smtcomp-current (2021)
8. cvc5 Authors: cvc5 user documentation. https://cvc5.github.io (2021)
9. Authors, C.: CVC4 SMT-COMP 2020 Single Query run script. https://github.com/CVC4/CVC4/blob/smtcomp2020/contrib/competitions/smt-comp/run-script-smtcomp-current (2020)
10. Backes, J., Berrueco, U., Bray, T., Brim, D., Cook, B., Gacek, A., Jhala, R., Luckow, K.S., McLaughlin, S., Menon, M., Peebles, D., Pugalia, U., Rungta, N., Schlesinger, C., Schodde, A., Tanuku, A., Varming, C., Viswanathan, D.: Stratified abstraction of access control policies. In: Lahiri, S.K., Wang, C. (eds.) Computer Aided Verification - 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21-24, 2020, Proceedings, Part I. Lecture Notes in Computer Science, vol. 12224, pp. 165–176. Springer (2020). https://doi.org/10.1007/978-3-030-53288-8_9
11. Backes, J., Bolignano, P., Cook, B., Dodge, C., Gacek, A., Luckow, K.S., Rungta, N., Tkachuk, O., Varming, C.: Semantic-based automated reasoning for AWS access policies using SMT. In: Bjørner, N., Gurfinkel, A. (eds.) 2018 Formal Methods in Computer Aided Design, FMCAD 2018, Austin, TX, USA, October 30 - November 2, 2018. pp. 1–9. IEEE (2018). https://doi.org/10.23919/FMCAD.2018.8602994

12. Bansal, K.: A branching heuristic in cvc4 smt solver. https://kshitij.io/articles/cvc4-branching-heuristic.pdf (2012)
13. Bansal, K., Barrett, C.W., Reynolds, A., Tinelli, C.: A new decision procedure for finite sets and cardinality constraints in SMT. CoRR **abs/1702.06259** (2017), http://arxiv.org/abs/1702.06259
14. Barbosa, H., Barrett, C., Brain, M., Kremer, G., Lachnitt, H., Mann, M., Mohamed, A., Mohamed, M.M.Y., Niemetz, A., Noetzli, A., Ozdemir, A., Preiner, M., Reynolds, A., Sheng, Y., Tinelli, C., Zohar, Y.: Artifact for Paper cvc5: A Versatile and Industrial-Strength SMT Solver (Nov 2021). https://doi.org/10.5281/zenodo.5740365, https://doi.org/10.5281/zenodo.5740365
15. Barbosa, H., Blanchette, J.C., Fleury, M., Fontaine, P.: Scalable fine-grained proofs for formula processing. Journal of Automated Reasoning **64**(3), 485–510 (2020). https://doi.org/10.1007/s10817-018-09502-y
16. Barbosa, H., Fontaine, P., Reynolds, A.: Congruence closure with free variables. In: Legay, A., Margaria, T. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 23rd International Conference, TACAS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings, Part II. Lecture Notes in Computer Science, vol. 10206, pp. 214–230 (2017). https://doi.org/10.1007/978-3-662-54580-5_13
17. Barbosa, H., Reynolds, A., Larraz, D., Tinelli, C.: Extending enumerative function synthesis via smt-driven classification. In: Barrett, C.W., Yang, J. (eds.) 2019 Formal Methods in Computer Aided Design, FMCAD 2019, San Jose, CA, USA, October 22-25, 2019. pp. 212–220. IEEE (2019). https://doi.org/10.23919/FMCAD.2019.8894267
18. Barbosa, H., Reynolds, A., Ouraoui, D.E., Tinelli, C., Barrett, C.W.: Extending SMT solvers to higher-order logic. In: Fontaine, P. (ed.) Proc. Conference on Automated Deduction (CADE). Lecture Notes in Computer Science, vol. 11716, pp. 35–54. Springer (2019). https://doi.org/10.1007/978-3-030-29436-6_3
19. Barnett, M., Chang, B.E., DeLine, R., Jacobs, B., Leino, K.R.M.: Boogie: A modular reusable verifier for object-oriented programs. In: de Boer, F.S., Bonsangue, M.M., Graf, S., de Roever, W.P. (eds.) Formal Methods for Components and Objects, 4th International Symposium, FMCO 2005, Amsterdam, The Netherlands, November 1-4, 2005, Revised Lectures. Lecture Notes in Computer Science, vol. 4111, pp. 364–387. Springer (2005). https://doi.org/10.1007/11804192_17
20. Barnett, M., Chang, B.Y.E., DeLine, R., Jacobs, B., Leino, K.R.M.: Boogie: A modular reusable verifier for object-oriented programs. In: International Symposium on Formal Methods for Components and Objects. pp. 364–387. Springer (2005)
21. Barrett, C., Conway, C.L., Deters, M., Hadarean, L., Jovanovic, D., King, T., Reynolds, A., Tinelli, C.: CVC4. In: CAV. Lecture Notes in Computer Science, vol. 6806, pp. 171–177. Springer (2011)
22. Barrett, C., Fontaine, P., Tinelli, C.: The Satisfiability Modulo Theories Library (SMT-LIB) (2020), http://smt-lib.org
23. Barrett, C., Nieuwenhuis, R., Oliveras, A., Tinelli, C.: Splitting on demand in SAT modulo theories. In: Hermann, M., Voronkov, A. (eds.) Proceedings of the 13th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR '06). Lecture Notes in Computer Science, vol. 4246, pp. 512–526. Springer-Verlag (Nov 2006), phnom Penh, Cambodia

24. Barrett, C., Shikanian, I., Tinelli, C.: An abstract decision procedure for a theory of inductive data types. JSAT **3**(1-2), 21–46 (2007). https://doi.org/10.3233/sat190028
25. Barrett, C., Stump, A., Tinelli, C.: The SMT-LIB Standard: Version 2.0. In: Gupta, A., Kroening, D. (eds.) Proceedings of the 8th International Workshop on Satisfiability Modulo Theories (Edinburgh, UK) (2010)
26. Barrett, C.W., Berezin, S.: CVC lite: A new implementation of the cooperating validity checker category B. In: Alur, R., Peled, D.A. (eds.) Computer Aided Verification (CAV). Lecture Notes in Computer Science, vol. 3114, pp. 515–518. Springer (2004). https://doi.org/10.1007/978-3-540-27813-9_49
27. Barrett, C.W., Tinelli, C.: CVC3. In: Damm, W., Hermanns, H. (eds.) Computer Aided Verification (CAV). Lecture Notes in Computer Science, vol. 4590, pp. 298–302. Springer (2007). https://doi.org/10.1007/978-3-540-73368-3_34
28. Beazley, D.M.: SWIG: an easy to use tool for integrating scripting languages with C and C++. In: Diekhans, M., Roseman, M. (eds.) Fourth Annual USENIX Tcl/Tk Workshop 1996, Monterey, California, USA, July 10-13, 1996. USENIX Association (1996), https://www.usenix.org/legacy/publications/library/proceedings/tcl96/beazley.html
29. Behnel, S., Bradshaw, R., Citro, C., Dalcin, L., Seljebotn, D.S., Smith, K.: Cython: The best of both worlds. Computing in Science & Engineering **13**(2), 31–39 (2011)
30. Bertot, Y., Castéran, P.: Interactive Theorem Proving and Program Development - Coq'Art: The Calculus of Inductive Constructions. Texts in Theoretical Computer Science. An EATCS Series, Springer (2004). https://doi.org/10.1007/978-3-662-07964-5
31. Biere, A., Fazekas, K., Fleury, M., Heisinger, M.: CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020. In: Balyo, T., Froleyks, N., Heule, M., Iser, M., Järvisalo, M., Suda, M. (eds.) Proc. of SAT Competition 2020 – Solver and Benchmark Descriptions. Department of Computer Science Report Series B, vol. B-2020-1, pp. 51–53. University of Helsinki (2020)
32. Blanchette, J.C., Böhme, S., Paulson, L.C.: Extending sledgehammer with SMT solvers. In: Bjørner, N., Sofronie-Stokkermans, V. (eds.) Automated Deduction - CADE-23 - 23rd International Conference on Automated Deduction, Wroclaw, Poland, July 31 - August 5, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6803, pp. 116–130. Springer (2011). https://doi.org/10.1007/978-3-642-22438-6_11
33. Bouchet, M., Cook, B., Cutler, B., Druzkina, A., Gacek, A., Hadarean, L., Jhala, R., Marshall, B., Peebles, D., Rungta, N., Schlesinger, C., Stephens, C., Varming, C., Warfield, A.: Block public access: trust safety verification of access control policies. In: Devanbu, P., Cohen, M.B., Zimmermann, T. (eds.) ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, November 8-13, 2020. pp. 281–291. ACM (2020). https://doi.org/10.1145/3368089.3409728
34. Bouton, T., de Oliveira, D.C.B., Déharbe, D., Fontaine, P.: veriT: An Open, Trustable and Efficient SMT-Solver. In: Schmidt, R.A. (ed.) Proc. Conference on Automated Deduction (CADE). Lecture Notes in Computer Science, vol. 5663, pp. 151–156. Springer (2009). https://doi.org/10.1007/978-3-642-02959-2_12
35. Bouton, T., Oliveira, D.C.B.D., Déharbe, D., Fontaine, P.: veriT: An open, trustable and efficient smt-solver. In: Schmidt, R.A. (ed.) Automated Deduc-

tion - CADE-22, 22nd International Conference on Automated Deduction, Montreal, Canada, August 2-7, 2009. Proceedings. Lecture Notes in Computer Science, vol. 5663, pp. 151–156. Springer (2009). https://doi.org/10.1007/978-3-642-02959-2_12
36. Brain, M., Niemetz, A., Preiner, M., Reynolds, A., Barrett, C.W., Tinelli, C.: Invertibility conditions for floating-point formulas. In: Dillig, I., Tasiran, S. (eds.) Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part II. Lecture Notes in Computer Science, vol. 11562, pp. 116–136. Springer (2019). https://doi.org/10.1007/978-3-030-25543-5_8
37. Brain, M., Schanda, F., Sun, Y.: Building better bit-blasting for floating-point problems. In: TACAS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings, Part I. LNCS, vol. 11427, pp. 79–98. Springer (2019). https://doi.org/10.1007/978-3-030-17462-0_5
38. Brillout, A., Kroening, D., Wahl, T.: Mixed abstractions for floating-point arithmetic. In: Proceedings of 9th International Conference on Formal Methods in Computer-Aided Design, FMCAD 2009, 15-18 November 2009, Austin, Texas, USA. pp. 69–76. IEEE (2009). https://doi.org/10.1109/FMCAD.2009.5351141
39. Bromberger, M., Weidenbach, C.: Fast cube tests for LIA constraint solving. In: IJCAR. Lecture Notes in Computer Science, vol. 9706, pp. 116–132. Springer (2016)
40. Cadar, C., Dunbar, D., Engler, D.R.: KLEE: unassisted and automatic generation of high-coverage tests for complex systems programs. In: Draves, R., van Renesse, R. (eds.) 8th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2008, December 8-10, 2008, San Diego, California, USA, Proceedings. pp. 209–224. USENIX Association (2008), http://www.usenix.org/events/osdi08/tech/full_papers/cadar/cadar.pdf
41. Cavada, R., Cimatti, A., Dorigatti, M., Griggio, A., Mariotti, A., Micheli, A., Mover, S., Roveri, M., Tonetta, S.: The nuxmv symbolic model checker. In: Biere, A., Bloem, R. (eds.) Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8559, pp. 334–342. Springer (2014). https://doi.org/10.1007/978-3-319-08867-9_22
42. Champion, A., Mebsout, A., Sticksel, C., Tinelli, C.: The kind 2 model checker. In: Chaudhuri, S., Farzan, A. (eds.) Computer Aided Verification - 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part II. Lecture Notes in Computer Science, vol. 9780, pp. 510–517. Springer (2016). https://doi.org/10.1007/978-3-319-41540-6_29
43. Christ, J., Hoenicke, J.: Weakly equivalent arrays. In: FroCos. Lecture Notes in Computer Science, vol. 9322, pp. 119–134. Springer (2015)
44. Christ, J., Hoenicke, J., Nutz, A.: Smtinterpol: An interpolating SMT solver. In: Donaldson, A.F., Parker, D. (eds.) Model Checking Software - 19th International Workshop, SPIN 2012, Oxford, UK, July 23-24, 2012. Proceedings. Lecture Notes in Computer Science, vol. 7385, pp. 248–254. Springer (2012). https://doi.org/10.1007/978-3-642-31759-0_19
45. Cimatti, A., Griggio, A., Irfan, A., Roveri, M., Sebastiani, R.: Invariant checking of NRA transition systems via incremental reduction to LRA with EUF. In: Legay, A., Margaria, T. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 23rd International Conference, TACAS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS

2017, Uppsala, Sweden, April 22-29, 2017, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10205, pp. 58–75 (2017). https://doi.org/10.1007/978-3-662-54577-5_4
46. Cimatti, A., Griggio, A., Schaafsma, B.J., Sebastiani, R.: The MathSAT5 SMT Solver. In: Proc. TACAS. Lecture Notes in Computer Science, vol. 7795, pp. 93–107. Springer (2013)
47. Cimatti, A., Griggio, A., Sebastiani, R.: Computing small unsatisfiable cores in satisfiability modulo theories. J. Artif. Intell. Res. (JAIR) **40**, 701–728 (2011). https://doi.org/10.1613/jair.3196
48. Cook, B.: Formal reasoning about the security of amazon web services. In: Chockler, H., Weissenbacher, G. (eds.) Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10981, pp. 38–47. Springer (2018). https://doi.org/10.1007/978-3-319-96145-3_3
49. Corbett, R.: Gnu bison (2021), https://www.gnu.org/software/bison/
50. Corzilius, F., Kremer, G., Junges, S., Schupp, S., Ábrahám, E.: SMT-RAT: an open source C++ toolbox for strategic and parallel SMT solving. In: Heule, M., Weaver, S.A. (eds.) Theory and Applications of Satisfiability Testing - SAT 2015 - 18th International Conference, Austin, TX, USA, September 24-27, 2015, Proceedings. Lecture Notes in Computer Science, vol. 9340, pp. 360–368. Springer (2015). https://doi.org/10.1007/978-3-319-24318-4_26
51. Craig, W.: Linear reasoning. A new form of the herbrand-gentzen theorem. J. Symb. Log. **22**(3), 250–268 (1957). https://doi.org/10.2307/2963593
52. Cuoq, P., Kirchner, F., Kosmatov, N., Prevosto, V., Signoles, J., Yakobowski, B.: Frama-c - A software analysis perspective. In: Eleftherakis, G., Hinchey, M., Holcombe, M. (eds.) Software Engineering and Formal Methods - 10th International Conference, SEFM 2012, Thessaloniki, Greece, October 1-5, 2012. Proceedings. Lecture Notes in Computer Science, vol. 7504, pp. 233–247. Springer (2012). https://doi.org/10.1007/978-3-642-33826-7_16
53. Detlefs, D., Nelson, G., Saxe, J.B.: Simplify: a theorem prover for program checking. J. ACM **52**(3), 365–473 (2005)
54. Dillig, I., Dillig, T., Aiken, A.: Cuts from proofs: a complete and practical technique for solving linear inequalities over integers. Formal Methods Syst. Des. **39**(3), 246–260 (2011)
55. Dutertre, B.: Yices 2.2. In: Biere, A., Bloem, R. (eds.) Computer Aided Verification (CAV). Lecture Notes in Computer Science, vol. 8559, pp. 737–744. Springer (2014). https://doi.org/10.1007/978-3-319-08867-9_49
56. Dutertre, B., de Moura, L.M.: A fast linear-arithmetic solver for DPLL(T). In: CAV. Lecture Notes in Computer Science, vol. 4144, pp. 81–94. Springer (2006)
57. Eén, N., Sörensson, N.: An extensible SAT-solver. In: SAT. Lecture Notes in Computer Science, vol. 2919, pp. 502–518. Springer (2003)
58. Ekici, B., Mebsout, A., Tinelli, C., Keller, C., Katz, G., Reynolds, A., Barrett, C.W.: Smtcoq: A plug-in for integrating SMT solvers into coq. In: Majumdar, R., Kuncak, V. (eds.) Computer Aided Verification (CAV). Lecture Notes in Computer Science, vol. 10427, pp. 126–133. Springer (2017). https://doi.org/10.1007/978-3-319-63390-9_7
59. England, M., Bradford, R.J., Davenport, J.H., Wilson, D.J.: Choosing a variable ordering for truth-table invariant cylindrical algebraic decomposition by incremental triangular decomposition. In: Hong, H., Yap, C. (eds.) Mathematical

Software - ICMS 2014 - 4th International Congress, Seoul, South Korea, August 5-9, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8592, pp. 450–457. Springer (2014). https://doi.org/10.1007/978-3-662-44199-2_68
60. Filliâtre, J., Paskevich, A.: Why3 - where programs meet provers. In: Felleisen, M., Gardner, P. (eds.) Programming Languages and Systems - 22nd European Symposium on Programming, ESOP 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24, 2013. Proceedings. Lecture Notes in Computer Science, vol. 7792, pp. 125–128. Springer (2013). https://doi.org/10.1007/978-3-642-37036-6_8
61. Ganesh, V., Dill, D.L.: A decision procedure for bit-vectors and arrays. In: Damm, W., Hermanns, H. (eds.) Computer Aided Verification, 19th International Conference, CAV 2007, Berlin, Germany, July 3-7, 2007, Proceedings. Lecture Notes in Computer Science, vol. 4590, pp. 519–531. Springer (2007). https://doi.org/10.1007/978-3-540-73368-3_52
62. Godefroid, P., Levin, M.Y., Molnar, D.A.: SAGE: whitebox fuzzing for security testing. Commun. ACM **55**(3), 40–44 (2012). https://doi.org/10.1145/2093548.2093564
63. Griggio, A.: An Effective SMT Engine for Formal Verification. Ph.D. thesis, University of Trento, Italy (2009)
64. Griggio, A.: A practical approach to satisfiability modulo linear integer arithmetic. Journal on Satisfiability, Boolean Modeling and Computation **8**(1-2), 1–27 (2012)
65. Gulwani, S., Jha, S., Tiwari, A., Venkatesan, R.: Synthesis of loop-free programs. In: Hall, M.W., Padua, D.A. (eds.) Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2011, San Jose, CA, USA, June 4-8, 2011. pp. 62–73. ACM (2011). https://doi.org/10.1145/1993498.1993506
66. Hadarean, L.: An efficient and trustworthy theory solver for bit-vectors in satisfiability modulo theories. Ph.D. thesis, Citeseer (2015)
67. Hadarean, L., Barrett, C.W., Reynolds, A., Tinelli, C., Deters, M.: Fine grained SMT proofs for the theory of fixed-width bit-vectors. In: Davis, M., Fehnker, A., McIver, A., Voronkov, A. (eds.) Logic for Programming, Artificial Intelligence, and Reasoning - 20th International Conference, LPAR-20 2015, Suva, Fiji, November 24-28, 2015, Proceedings. Lecture Notes in Computer Science, vol. 9450, pp. 340–355. Springer (2015). https://doi.org/10.1007/978-3-662-48899-7_24
68. Hajdu, Á., Jovanovic, D.: solc-verify: A modular verifier for solidity smart contracts. In: Chakraborty, S., Navas, J.A. (eds.) Verified Software. Theories, Tools, and Experiments - 11th International Conference, VSTTE 2019, New York City, NY, USA, July 13-14, 2019, Revised Selected Papers. Lecture Notes in Computer Science, vol. 12031, pp. 161–179. Springer (2019). https://doi.org/10.1007/978-3-030-41600-3_11
69. Harper, R., Honsell, F., Plotkin, G.: A Framework for Defining Logics. Journal of the Association for Computing Machinery **40**(1), 143–184 (Jan 1993)
70. Hauzar, D., Marché, C., Moy, Y.: Counterexamples from proof failures in SPARK. In: Nicola, R.D., eva Kühn (eds.) Software Engineering and Formal Methods - 14th International Conference, SEFM 2016, Held as Part of STAF 2016, Vienna, Austria, July 4-8, 2016, Proceedings. Lecture Notes in Computer Science, vol. 9763, pp. 215–233. Springer (2016). https://doi.org/10.1007/978-3-319-41591-8_15
71. Hughes, R.J.M.: Super combinators: a new implementation method for applicative languages. In: Symposium on LISP and Functional Programming. pp. 1–10 (1982)

72. Hyvärinen, A.E.J., Marescotti, M., Alt, L., Sharygina, N.: OpenSMT2: An SMT solver for multi-core and cloud computing. In: Creignou, N., Berre, D.L. (eds.) Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings. Lecture Notes in Computer Science, vol. 9710, pp. 547–553. Springer (2016). https://doi.org/10.1007/978-3-319-40970-2_35
73. Iosif, R., Serban, C., Reynolds, A., Sighireanu, M.: Encoding separation logic in smt-lib v2.5 (2018)
74. Jovanovic, D., Barrett, C.W.: Polite theories revisited. In: LPAR (Yogyakarta). Lecture Notes in Computer Science, vol. 6397, pp. 402–416. Springer (2010)
75. Jovanovic, D., Barrett, C.W.: Being careful about theory combination. Formal Methods Syst. Des. **42**(1), 67–90 (2013)
76. Jovanovic, D., Dutertre, B.: Libpoly: A library for reasoning about polynomials. In: Brain, M., Hadarean, L. (eds.) Proceedings of the 15th International Workshop on Satisfiability Modulo Theories affiliated with the International Conference on Computer-Aided Verification (CAV 2017), Heidelberg, Germany, July 22 - 23, 2017. CEUR Workshop Proceedings, vol. 1889, pp. 28–39. CEUR-WS.org (2017), http://ceur-ws.org/Vol-1889/paper3.pdf
77. Katz, G., Barrett, C.W., Tinelli, C., Reynolds, A., Hadarean, L.: Lazy proofs for dpll(t)-based SMT solvers. In: Piskac, R., Talupur, M. (eds.) 2016 Formal Methods in Computer-Aided Design, FMCAD 2016, Mountain View, CA, USA, October 3-6, 2016. pp. 93–100. IEEE (2016). https://doi.org/10.1109/FMCAD.2016.7886666
78. King, T.: Effective Algorithms for the Satisfiability of Quantifier-Free Formulas Over Linear Real and Integer Arithmetic. Ph.D. thesis, New York University (2014)
79. King, T., Barrett, C.W., Dutertre, B.: Simplex with sum of infeasibilities for SMT. In: Formal Methods in Computer-Aided Design, FMCAD 2013, Portland, OR, USA, October 20-23, 2013. pp. 189–196. IEEE (2013), https://ieeexplore.ieee.org/document/6679409/
80. King, T., Barrett, C.W., Tinelli, C.: Leveraging linear and mixed integer programming for SMT. In: Formal Methods in Computer-Aided Design, FMCAD 2014, Lausanne, Switzerland, October 21-24, 2014. pp. 139–146. IEEE (2014). https://doi.org/10.1109/FMCAD.2014.6987606
81. Leino, K.M.: Accessible software verification with dafny. IEEE Software **34**(06), 94–97 (nov 2017). https://doi.org/10.1109/MS.2017.4121212
82. Leino, K.R.M.: Dafny: An automatic program verifier for functional correctness. In: Clarke, E.M., Voronkov, A. (eds.) Logic for Programming, Artificial Intelligence, and Reasoning - 16th International Conference, LPAR-16, Dakar, Senegal, April 25-May 1, 2010, Revised Selected Papers. Lecture Notes in Computer Science, vol. 6355, pp. 348–370. Springer (2010). https://doi.org/10.1007/978-3-642-17511-4_20
83. Liang, S.: The Java Native interface : programmer's guide and specification / Sheng Liang. Java series, Addison-Wesley, Reading, Mass. ; Harlow, England (1999)
84. Liang, T., Reynolds, A., Tinelli, C., Barrett, C.W., Deters, M.: A DPLL(T) theory solver for a theory of strings and regular expressions. In: CAV. Lecture Notes in Computer Science, vol. 8559, pp. 646–662. Springer (2014)
85. Liang, T., Tsiskaridze, N., Reynolds, A., Tinelli, C., Barrett, C.W.: A decision procedure for regular membership and length constraints over unbounded strings.

In: FroCos. Lecture Notes in Computer Science, vol. 9322, pp. 135–150. Springer (2015)
86. Mattarei, C., Mann, M., Barrett, C.W., Daly, R.G., Huff, D., Hanrahan, P.: Cosa: Integrated verification for agile hardware design. In: Bjørner, N., Gurfinkel, A. (eds.) 2018 Formal Methods in Computer Aided Design, FMCAD 2018, Austin, TX, USA, October 30 - November 2, 2018. pp. 1–5. IEEE (2018). https://doi.org/10.23919/FMCAD.2018.8603014
87. Meng, B., Reynolds, A., Tinelli, C., Barrett, C.W.: Relational constraint solving in SMT. In: de Moura, L. (ed.) Automated Deduction - CADE 26 - 26th International Conference on Automated Deduction, Gothenburg, Sweden, August 6-11, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10395, pp. 148–165. Springer (2017). https://doi.org/10.1007/978-3-319-63046-5_10
88. de Moura, L., Ullrich, S.: The lean 4 theorem prover and programming language. In: Platzer, A., Sutcliffe, G. (eds.) Automated Deduction - CADE 28 - 28th International Conference on Automated Deduction, Virtual Event, July 12-15, 2021, Proceedings. Lecture Notes in Computer Science, vol. 12699, pp. 625–635. Springer (2021). https://doi.org/10.1007/978-3-030-79876-5_37
89. de Moura, L.M., Bjørner, N.: Efficient e-matching for SMT solvers. In: Pfenning, F. (ed.) Automated Deduction - CADE-21, 21st International Conference on Automated Deduction, Bremen, Germany, July 17-20, 2007, Proceedings. Lecture Notes in Computer Science, vol. 4603, pp. 183–198. Springer (2007). https://doi.org/10.1007/978-3-540-73595-3_13
90. de Moura, L.M., Bjørner, N.: Z3: an efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings. Lecture Notes in Computer Science, vol. 4963, pp. 337–340. Springer (2008). https://doi.org/10.1007/978-3-540-78800-3_24, https://doi.org/10.1007/978-3-540-78800-3_24
91. de Moura, L.M., Bjørner, N.: Generalized, efficient array decision procedures. In: FMCAD. pp. 45–52. IEEE (2009)
92. Niemetz, A., Preiner, M.: Bitwuzla at the SMT-COMP 2020. CoRR **abs/2006.01621** (2020), https://arxiv.org/abs/2006.01621
93. Niemetz, A., Preiner, M.: Ternary propagation-based local search for more bit-precise reasoning. In: 2020 Formal Methods in Computer Aided Design, FMCAD 2020, Haifa, Israel, September 21-24, 2020. pp. 214–224. IEEE (2020). https://doi.org/10.34727/2020/isbn.978-3-85448-042-6_29
94. Niemetz, A., Preiner, M., Biere, A.: Propagation based local search for bit-precise reasoning. Formal Methods Syst. Des. **51**(3), 608–636 (2017). https://doi.org/10.1007/s10703-017-0295-6
95. Niemetz, A., Preiner, M., Reynolds, A., Barrett, C.W., Tinelli, C.: On solving quantified bit-vector constraints using invertibility conditions. Formal Methods Syst. Des. **57**(1), 87–115 (2021). https://doi.org/10.1007/s10703-020-00359-9
96. Niemetz, A., Preiner, M., Reynolds, A., Barrett, C.W., Tinelli, C.: Syntax-guided quantifier instantiation. In: Groote, J.F., Larsen, K.G. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 27th International Conference, TACAS 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27 - April 1, 2021, Proceedings, Part II. Lecture Notes in Computer Science, vol. 12652, pp. 145–163. Springer (2021). https://doi.org/10.1007/978-3-030-72013-1_8

97. Niemetz, A., Preiner, M., Reynolds, A., Zohar, Y., Barrett, C.W., Tinelli, C.: Towards bit-width-independent proofs in SMT solvers. In: Fontaine, P. (ed.) Automated Deduction - CADE 27 - 27th International Conference on Automated Deduction, Natal, Brazil, August 27-30, 2019, Proceedings. Lecture Notes in Computer Science, vol. 11716, pp. 366–384. Springer (2019). https://doi.org/10.1007/978-3-030-29436-6_22
98. Niemetz, A., Preiner, M., Wolf, C., Biere, A.: Btor2 , btormc and boolector 3.0. In: Chockler, H., Weissenbacher, G. (eds.) Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10981, pp. 587–595. Springer (2018). https://doi.org/10.1007/978-3-319-96145-3_32
99. Nieuwenhuis, R., Oliveras, A., Tinelli, C.: Solving SAT and SAT modulo theories: From an abstract davis–putnam–logemann–loveland procedure to dpll(T). J. ACM **53**(6), 937–977 (2006). https://doi.org/10.1145/1217856.1217859
100. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL - A Proof Assistant for Higher-Order Logic, Lecture Notes in Computer Science, vol. 2283. Springer (2002). https://doi.org/10.1007/3-540-45949-9
101. Nötzli, A., Reynolds, A., Barbosa, H., Niemetz, A., Preiner, M., Barrett, C.W., Tinelli, C.: Syntax-guided rewrite rule enumeration for SMT solvers. In: Janota, M., Lynce, I. (eds.) Theory and Applications of Satisfiability Testing - SAT 2019 - 22nd International Conference, SAT 2019, Lisbon, Portugal, July 9-12, 2019, Proceedings. Lecture Notes in Computer Science, vol. 11628, pp. 279–297. Springer (2019). https://doi.org/10.1007/978-3-030-24258-9_20
102. Organizers, S.C.: SMT-COMP 2021. https://smt-comp.github.io/2021/ (2021)
103. Organizers, S.C.: SyGuS-Comp 2019. https://sygus.org/comp/2019/ (2021)
104. Padhi, S., Polgreen, E., Raghothaman, M., Reynolds, A., Udupa, A.: The sygus language standard version 2.1 (2021)
105. Parr, T.: ANTLRv3 (2021), https://www.antlr3.org/
106. Paxson, V.: Flex lexical analyser generator (2021), https://github.com/westes/flex
107. Piskac, R., Kuncak, V.: Decision procedures for multisets with cardinality constraints. In: Logozzo, F., Peled, D.A., Zuck, L.D. (eds.) Verification, Model Checking, and Abstract Interpretation, 9th International Conference, VMCAI 2008, San Francisco, USA, January 7-9, 2008, Proceedings. Lecture Notes in Computer Science, vol. 4905, pp. 218–232. Springer (2008). https://doi.org/10.1007/978-3-540-78163-9_20
108. Ranise, S., Ringeissen, C., Zarba, C.G.: Combining data structures with nonstably infinite theories using many-sorted logic. In: FroCoS. Lecture Notes in Computer Science, vol. 3717, pp. 48–64. Springer (2005)
109. Reynolds, A., Barbosa, H., Fontaine, P.: Revisiting enumerative instantiation. In: Beyer, D., Huisman, M. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 24th International Conference, TACAS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings, Part II. Lecture Notes in Computer Science, vol. 10806, pp. 112–131. Springer (2018). https://doi.org/10.1007/978-3-319-89963-3_7
110. Reynolds, A., Barbosa, H., Larraz, D., Tinelli, C.: Scalable algorithms for abduction via enumerative syntax-guided synthesis. In: Peltier, N., Sofronie-Stokkermans, V. (eds.) Automated Reasoning - 10th International Joint Con-

ference, IJCAR 2020, Paris, France, July 1-4, 2020, Proceedings, Part I. Lecture Notes in Computer Science, vol. 12166, pp. 141–160. Springer (2020). https://doi.org/10.1007/978-3-030-51074-9_9
111. Reynolds, A., Barbosa, H., Nötzli, A., Barrett, C.W., Tinelli, C.: cvc4sy: Smart and fast term enumeration for syntax-guided synthesis. In: Dillig, I., Tasiran, S. (eds.) Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part II. Lecture Notes in Computer Science, vol. 11562, pp. 74–83. Springer (2019). https://doi.org/10.1007/978-3-030-25543-5_5
112. Reynolds, A., Blanchette, J.C.: A decision procedure for (co)datatypes in SMT solvers. In: Felty, A.P., Middeldorp, A. (eds.) Automated Deduction - CADE-25 - 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings. Lecture Notes in Computer Science, vol. 9195, pp. 197–213. Springer (2015). https://doi.org/10.1007/978-3-319-21401-6_13
113. Reynolds, A., Blanchette, J.C., Cruanes, S., Tinelli, C.: Model finding for recursive functions in SMT. In: Olivetti, N., Tiwari, A. (eds.) Automated Reasoning - 8th International Joint Conference, IJCAR 2016, Coimbra, Portugal, June 27 - July 2, 2016, Proceedings. Lecture Notes in Computer Science, vol. 9706, pp. 133–151. Springer (2016). https://doi.org/10.1007/978-3-319-40229-1_10
114. Reynolds, A., Deters, M., Kuncak, V., Tinelli, C., Barrett, C.W.: Counterexample-guided quantifier instantiation for synthesis in SMT. In: Kroening, D., Pasareanu, C.S. (eds.) Computer Aided Verification - 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part II. Lecture Notes in Computer Science, vol. 9207, pp. 198–216. Springer (2015). https://doi.org/10.1007/978-3-319-21668-3_12
115. Reynolds, A., Iosif, R., Serban, C., King, T.: A decision procedure for separation logic in SMT. In: Artho, C., Legay, A., Peled, D. (eds.) Automated Technology for Verification and Analysis - 14th International Symposium, ATVA 2016, Chiba, Japan, October 17-20, 2016, Proceedings. Lecture Notes in Computer Science, vol. 9938, pp. 244–261 (2016). https://doi.org/10.1007/978-3-319-46520-3_16
116. Reynolds, A., King, T., Kuncak, V.: Solving quantified linear arithmetic by counterexample-guided instantiation. Formal Methods Syst. Des. **51**(3), 500–532 (2017). https://doi.org/10.1007/s10703-017-0290-y
117. Reynolds, A., Kuncak, V.: Induction for SMT solvers. In: D'Souza, D., Lal, A., Larsen, K.G. (eds.) Verification, Model Checking, and Abstract Interpretation - 16th International Conference, VMCAI 2015, Mumbai, India, January 12-14, 2015. Proceedings. Lecture Notes in Computer Science, vol. 8931, pp. 80–98. Springer (2015). https://doi.org/10.1007/978-3-662-46081-8_5
118. Reynolds, A., Nötzli, A., Barrett, C.W., Tinelli, C.: High-level abstractions for simplifying extended string constraints in SMT. In: CAV (2). Lecture Notes in Computer Science, vol. 11562, pp. 23–42. Springer (2019)
119. Reynolds, A., Nötzli, A., Barrett, C.W., Tinelli, C.: A decision procedure for string to code point conversion. In: IJCAR (1). Lecture Notes in Computer Science, vol. 12166, pp. 218–237. Springer (2020)
120. Reynolds, A., Nötzli, A., Barrett, C.W., Tinelli, C.: Reductions for strings and regular expressions revisited. In: FMCAD. pp. 225–235. IEEE (2020)
121. Reynolds, A., Tinelli, C., Goel, A., Krstic, S.: Finite model finding in SMT. In: Sharygina, N., Veith, H. (eds.) Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings. Lecture Notes in Computer Science, vol. 8044, pp. 640–655. Springer (2013). https://doi.org/10.1007/978-3-642-39799-8_42

122. Reynolds, A., Tinelli, C., Goel, A., Krstic, S., Deters, M., Barrett, C.W.: Quantifier instantiation techniques for finite model finding in SMT. In: Bonacina, M.P. (ed.) Automated Deduction - CADE-24 - 24th International Conference on Automated Deduction, Lake Placid, NY, USA, June 9-14, 2013. Proceedings. Lecture Notes in Computer Science, vol. 7898, pp. 377–391. Springer (2013). https://doi.org/10.1007/978-3-642-38574-2_26
123. Reynolds, A., Tinelli, C., Jovanovic, D., Barrett, C.W.: Designing theory solvers with extensions. In: Dixon, C., Finger, M. (eds.) Frontiers of Combining Systems - 11th International Symposium, FroCoS 2017, Brasília, Brazil, September 27-29, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10483, pp. 22–40. Springer (2017). https://doi.org/10.1007/978-3-319-66167-4_2
124. Reynolds, A., Tinelli, C., de Moura, L.M.: Finding conflicting instances of quantified formulas in SMT. In: Formal Methods in Computer-Aided Design, FMCAD 2014, Lausanne, Switzerland, October 21-24, 2014. pp. 195–202. IEEE (2014). https://doi.org/10.1109/FMCAD.2014.6987613
125. Reynolds, A., Viswanathan, A., Barbosa, H., Tinelli, C., Barrett, C.W.: Datatypes with shared selectors. In: Galmiche, D., Schulz, S., Sebastiani, R. (eds.) Automated Reasoning - 9th International Joint Conference, IJCAR 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings. Lecture Notes in Computer Science, vol. 10900, pp. 591–608. Springer (2018). https://doi.org/10.1007/978-3-319-94205-6_39
126. Reynolds, A., Woo, M., Barrett, C.W., Brumley, D., Liang, T., Tinelli, C.: Scaling up DPLL(T) string solvers using context-dependent simplification. In: CAV (2). Lecture Notes in Computer Science, vol. 10427, pp. 453–474. Springer (2017)
127. Schkufza, E., Sharma, R., Aiken, A.: Stochastic program optimization. Commun. ACM **59**(2), 114–122 (2016). https://doi.org/10.1145/2863701
128. Schurr, H., Fleury, M., Barbosa, H., Fontaine, P.: Alethe: Towards a generic SMT proof format (extended abstract). In: Keller, C., Fleury, M. (eds.) Workshop on Proof eXchange for Theorem Proving (PxTP). EPTCS, vol. 336, pp. 49–54 (2021). https://doi.org/10.4204/EPTCS.336.6, https://doi.org/10.4204/EPTCS.336.6
129. Schurr, H., Fleury, M., Desharnais, M.: Reliable reconstruction of fine-grained proofs in a proof assistant. In: Platzer, A., Sutcliffe, G. (eds.) Proc. Conference on Automated Deduction (CADE). Lecture Notes in Computer Science, vol. 12699, pp. 450–467. Springer (2021). https://doi.org/10.1007/978-3-030-79876-5_26
130. Sheng, Y., Zohar, Y., Ringeissen, C., Lange, J., Fontaine, P., Barrett, C.W.: Politeness for the theory of algebraic datatypes. In: Peltier, N., Sofronie-Stokkermans, V. (eds.) Automated Reasoning - 10th International Joint Conference, IJCAR 2020, Paris, France, July 1-4, 2020, Proceedings, Part I. Lecture Notes in Computer Science, vol. 12166, pp. 238–255. Springer (2020). https://doi.org/10.1007/978-3-030-51074-9_14
131. Soos, M.: CryptoMiniSat. https://github.com/msoos/cryptominisat (2020)
132. Stump, A., Barrett, C.W., Dill, D.L.: CVC: A cooperating validity checker. In: Brinksma, E., Larsen, K.G. (eds.) Computer Aided Verification (CAV). Lecture Notes in Computer Science, vol. 2404, pp. 500–504. Springer (2002). https://doi.org/10.1007/3-540-45657-0_40
133. Stump, A., Oe, D., Reynolds, A., Hadarean, L., Tinelli, C.: SMT proof checking using a logical framework. Formal Methods in System Design **42**(1), 91–118 (2013). https://doi.org/10.1007/s10703-012-0163-3
134. Sutcliffe, G.: The TPTP Problem Library and Associated Infrastructure. From CNF to TH0, TPTP v6.4.0. Journal of Automated Reasoning **59**(4), 483–502 (2017)

135. Tillmann, N., de Halleux, J.: Pex-white box test generation for .net. In: Beckert, B., Hähnle, R. (eds.) Tests and Proofs - 2nd International Conference, TAP 2008, Prato, Italy, April 9-11, 2008. Proceedings. Lecture Notes in Computer Science, vol. 4966, pp. 134–153. Springer (2008). https://doi.org/10.1007/978-3-540-79124-9_10
136. Trentin, P.: Optimization Modulo Theories with OptiMathSAT. Ph.D. thesis, University of Trento (2019)
137. Zhong, J.E., Cheang, K., Qadeer, S., Grieskamp, W., Blackshear, S., Park, J., Zohar, Y., Barrett, C.W., Dill, D.L.: The move prover. In: Lahiri, S.K., Wang, C. (eds.) Computer Aided Verification - 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21-24, 2020, Proceedings, Part I. Lecture Notes in Computer Science, vol. 12224, pp. 137–150. Springer (2020). https://doi.org/10.1007/978-3-030-53288-8_7
138. Zohar, Y., Irfan, A., Mann, M., Niemetz, A., Nötzli, A., Preiner, M., Reynolds, A., Barrett, C., Tinelli, C.: Bit-Precise Reasoning via Int-Blasting, to appear in the proceedings of VMCAI 2022

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.